LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data

Wouter Beek[1], Laurens Rietveld[1], Hamid R. Bazoobandi[1], Jan Wielemaker[1], and Stefan Schlobach[1]

Dept. of Computer Science, VU University Amsterdam, NL
{w.g.j.beek,laurens.rietveld,h.bazoubandi,j.wielemaker,k.s.schlobach}@vu.nl

Abstract. It is widely accepted that *proper* data publishing is difficult. The majority of Linked Open Data (LOD) does not meet even a core set of data publishing guidelines. Moreover, datasets that are *clean* at creation, can get *stains* over time. As a result, the LOD cloud now contains a high level of *dirty* data that is difficult for humans to clean and for machines to process.
Existing solutions for cleaning data (standards, guidelines, tools) are targeted towards human data creators, who can (and do) choose not to use them. This paper presents the LOD Laundromat which removes stains from data without any human intervention. This fully automated approach is able to make very large amounts of LOD more easily available for further processing *right now*.
LOD Laundromat is not a new dataset, but rather a uniform point of entry to a collection of cleaned siblings of existing datasets. It provides researchers and application developers a wealth of data that is guaranteed to conform to a specified set of best practices, thereby greatly improving the chance of data actually being (re)used.

Keywords: Data Publishing, Data Cleaning, Data Reuse, Standards Conformance

1 Introduction

Uptake of Linked Open Data (LOD) has seen a tremendous growth over the last decade. Due to the inherently heterogeneous nature of interlinked datasets that come from very different sources, LOD is not only a fertile environment for innovative data (re)use, but also for mistakes and incompatibilities [4, 5]. Such stains in datasets not only degrade a dataset's own quality, but also the quality of other datasets that link to it (e.g., by using `owl:sameAs`). There is thus an incentive to clean stains in LOD that goes beyond that of the original dataset creators.

Existing solutions for cleaning Semantic Web (SW) data (standards, guidelines, tools) are targeted towards human data creators, who can (and do) choose not to use them. Therefore, despite these efforts, much of LOD is still difficult to use today, mostly because of mistakes for which solutions exist. This poses an unnecessary impediment to the (re)use of LOD for academic and commercial purposes.

This paper presents the LOD Laundromat, which takes *immediate* action by targeting the *data* directly, not its maintainers. By cleaning stains in LOD without any human intervention, LOD Laundromat is able to make very large amounts of LOD more easily available for further processing *right now*. The collection of cleaned datasets that LOD Laundromat produces are standards- and guidelines-compliant siblings of existing, idiosyncratic datasets.

The data-oriented approach of LOD Laundromat is complementary to existing efforts, since it is preferable that someday the original dataset is cleaned by its own maintainers. However, we believe that until that day, our complementary approach is necessary to make LOD succeed while the momentum is still there. LOD Laundromat is unlike any of the existing initiatives towards realizing standards-compliant LOD in each of the following three ways:

1. The **scale** on which clean data is made available: LOD Laundromat comprises thousands of data files, and billions of triples.
2. The **speed** at which data is cleaned and made available: LOD Laundromat cleans about a billion triples a day and makes them immediately available online.
3. The **level of automation**. LOD Laundromat automates the entire data processing pipeline, from dataset discovery to serialization in a standards-compliant canonical format that enables easy reuse.

Besides making LOD standards-compliant, LOD Laundromat implements existing standards in such a way that the resultant data documents are specifically geared towards easy reuse by further tooling. This includes simplifying certain aspects of LOD that often cause problems in practice, such as blank nodes, and significantly reducing the complexity for post-processors to parse the data, e.g., through a syntax that is regular expression-friendly.

The LOD Laundromat is available at `http://lodlaundromat.org`. The collection of datasets that it comprises is continuously being extended. Anyone can add new seed points to the LOD Laundry Basket by using a Web form or HTTP GET request. The fully automated LOD Washing Machine takes seed points from the LOD Laundry Basket and cleans them. Cleaned datasets are disseminated in the LOD Wardrobe. Human data consumers are able to navigate a large collection of high-quality datasets. Machine processors are able to easily load very large amounts of real-world data, by selecting clean data documents through a SPARQL query. For illustrative purposes, various visualizations about the cleaned data are available as well.

This paper is organized as follows: section 2 gives an overview of related work. Section 3 specifies the requirements we pose for clean and useful data, and briefly explores alternative approaches towards collecting large amounts of Linked Data. Section 4 details the major operationalization decisions that allow the data cleaning process to be fully automated. Section 5 elaborates on the way in which LOD Laundromat makes data available for further processing. Section 6 concludes and mentions future work.

2 Related Work

In this section we firstly discuss standards and best practices with respect to Linked Data publishing. Secondly, we discuss existing Linked Data collections and crawlers. Finally, we discuss available Linked Data catalogs and their advantages and disadvantages.

2.1 Standards

The VoID standard[1] is a vocabulary to formally describe datasets. It supports general metadata (e.g., the homepage of a dataset), access metadata (e.g., which protocols are available), possible links with other datasets, as well as structural metadata. Structural metadata includes exemplary resources and statistics (e.g., the number of triples, properties and classes).

Bio2RDF [2] presents a collection of dataset metrics that extends the structural metadata of the VoID description, and provides more detail (e.g. the number of unique objects linked from each predicate).

While such standards are useful from both the data publisher and the data consumer perspective, uptake of VoID is lacking.[2] Additionally, from a data consumer perspective, the issue of findability through fully automated means is not resolved.

A number of observations and statistics related to Linked Data publishing best practices are presented in [3] and by the W3C Linked Data best practices working group[3]. In addition, [5] have analyzed over a billion triples from 4 million crawled RDF/XML documents. This analysis shows that on average 15.7% of the RDF nodes are blank nodes. Furthermore, their analysis shows that most Linked Data is not fully standards-compliant, corroborating the need for sanitizing Linked Data. However, note that this study is purely observational, and the accessed data is not made available in a cleaned form.

2.2 Data Crawlers

Sindice [8] presents itself as a Semantic Web indexer. The main question Sindice tries to address, is how and where to find statements about certain resources. It does so by crawling Linked Data resources, including RDF, RDFa and Microformats, although large RDF datasets are imported on a per-instance and manual opt-in basis. Sindice maintains a large cache of this data, and provides access through a user interface and API. Public access to the raw data crawler by Sindice is not available, nor is access through SPARQL, restricting the usefulness of Sindice for Semantic Web and Big Data research. Built on top of Sindice, Sig.ma [10] is an explorative interactive tool, that enables Linked Data discovery.

[1] `http://www.w3.org/TR/void/`

[2] A overview of VoID descriptions that can be found by automated means, is given by the SPARQL Endpoint Status service: `http://sparqles.okfn.org/`

[3] `http://www.w3.org/TR/ld-bp/`

Similar to Sindice, Sig.ma provides an extensive user interface, as well as API access. Even though this service can be quite useful for data exploration, the actual, raw data is not accessible for further processing.

Contrary to Sig.ma and Sindice, data from the Billion Triple Challenge[4] (BTC) 2012 are publicly available and are often used in Big Data research. The BTC dataset is crawled from the LOD cloud[5], and consists of 1.4 billion triples. It includes large RDF datasets, as well as data in RDFa and Microformats. However, this dataset is not a complete crawl of the Linked Open Data cloud (nor does it aim to be), as datasets from several catalogs are missing from the BTC. Additionally, the latest version of this dataset dates back to 2012.

Freebase [1] publishes 1.9 billion triples, taken from manual user input and existing RDF and Microformat datasets. Access to Freebase is possible through an API, through a (non-SPARQL) structured query language, and as a complete dump of N-Triples. However, these dumps include many non-conformant, syntactically incorrect triples. To give a concrete example, the data file that is the dereference of the Freebase concept 'Monkey'[6] visually appears to contain hundreds of triples, but a state-of-the-art standards-conformant parser such as Raptor[7] only extracts 30 triples. Additionally, knowing *which* datasets are included in Freebase, and finding these particular datasets, is not trivial.

Similarly, LODCache[8], provided by OpenLink, takes a similar crawling approach as Freebase does, but does not make a data dump available, making actual re-use of the data difficult. However, LODCache does have a SPARQL endpoint, as well as features such as entity URI and label lookup.

The Open Data Communities service[9] is the UK Department for Communities and Local Government's official Linked Open Data site. These datasets are published as data dumps, and are accessible through SPARQL and API calls. Although this service supports a broad selection of protocols for accessing the data, the number of datasets is limited and restricted to a particular domain.

Finally, DyLDO [6] is a long-term experiment to monitor the dynamics of a core set of 80 thousand Linked Data documents on a weekly basis. Each week's crawl is published as an N-Quads file. This work provides interesting insight in how Linked Data evolves over time. However, it is not possible to easily select the triples from a *single* dataset, and not all datasets belonging to the LOD cloud are included. Another form of incompleteness stems from the fact that the crawl is based on URI dereferences, not guaranteeing that a dataset is included in its entirety (see section 3).

[4] `http://km.aifb.kit.edu/projects/btc-2012/`

[5] `http://lod-cloud.net/`

[6] `http://rdf.freebase.com/ns/m.08pbxl`

[7] Tested with version 2.0.9, `http://librdf.org/raptor/rapper.html`

[8] `http://lod.openlinksw.com/`

[9] `http://opendatacommunities.org/`

2.3 Portals

Several Linked Data portals exist, that try to improve the findability of Linked Datasets. The Datahub[10] lists a large set of RDF datasets and SPARQL endpoints, including the famous collection of datasets that is called the LOD cloud. Datasets that are missing from the BTC collection are present in the Datahub catalog, and the other way round. This catalog is updated manually, and there is no direct connection to the data: all metadata comes from user input. This increases the risk of stale dataset descriptions[11] and missing or incorrect metadata. vocab.cc [9] builds on top of the BTC dataset. At the time of writing, it provides a list of 422 vocabularies. Access to these vocabularies is possible throguh SPARQL and an API. This service increases the ease of finding and re-using existing vocabularies. It has the same incompleteness properties that the BTC has, and does not (intend to) include instance data.

3 Context

Due to the points mentioned above, the poor data quality on the LOD cloud poses great challenges to Big Data and SW researchers, as well as to the developers of Web-scale applications and services. In practice, this means that LOD is less effectively (re)used than it should and could be. We first enumerate the requirements that we pose on clean datasets in order to be easily (re)usable (section 3.1). We then compare three approaches towards collecting LOD, and evaluate each with respect to the completeness of their results (section 3.2).

3.1 Requirements

Besides the obvious requirements of being syntactically correct and standards-compliant, we also pose additional requirements for how SW datasets should be serialized and disseminated. We enumerate these additional requirements, and briefly explain why they result in data that is more useful for Big Data researchers and LOD developers in practice.

Easy grammar We want LOD to be disseminated in such a way that it is easy to handle by subsequent processors. These subsequent processors are often non-RDF tools, such as Pig [7], grep, sed, and the like. Such easy post-processing is guaranteed by adherence to a uniform data format that can be safely parsed in an unambiguous way, e.g., by being able to extract triples and terms with one simple regular expression.

Speed We want to allow tools to process LOD in a speedy way. Parsing of data documents may be slow due to the use of inefficient serialization formats (e.g., RDF/XML, RDFa), the occurrence of large numbers of duplicate triples, or the presence of syntax errors that necessitate a parser to come up with fallback options.

[10] `http://datahub.io/`

[11] For example, DBpedia links to version 3.5.1 instead of 3.9: `http://datahub.io/dataset/dbpedia` (12 May 2014)

Quantity We want to make available a large number of data documents (tens of thousands) and triples (billions), to cover a large parts of the LOD cloud.

Combine We want to make it easy to combine data documents, e.g., splitting a single document into multiple ones, or appending multiple documents into a single one. This is important for load job balancing in large-scale processing, since the distribution of triples across data documents is otherwise very uneven.

Streaming We want to support streamed processing of triples, in such a way that the streamed processor does not have to perform additional bookkeeping on the processed data, e.g., having to check for statements that were already observed earlier.

Completeness The data must be a complete representation of the input dataset, to the extent at which the original dataset is standards-compliant.

3.2 Dataset completeness

The first problem that we come across when collecting large amounts of LOD, is that it is difficult to claim completeness while collecting LOD. Since there are alternative approaches towards collecting large volumes of LOD, we give an overview of the incompleteness issues that arise for each of those alternatives. At the moment, three options exist for collecting large volumes of LOD:

1. Crawling resources
2. Querying endpoints
3. Downloading datadumps

Resource crawlers use the dereferenceability of IRIs in order to find LOD. This approach has the following deficiencies:

1. Datasets that do not contain dereferenceable IRIs are ignored. In [4], 7.2% of the crawled IRIs were not dereferenceable.
2. For IRIs that can be dereferenced, back-links are often not included [5] As a consequence of this, even datasets that contain dereferenceable IRIs exclusively can still have parts that cannot be reached by a crawler.
3. Even for datasets that have only dereferenceable IRIs that include back-links, the crawler can never be certain that the entire dataset has been crawled.

Querying endpoints provides another way of collecting large volumes of LOD. The disadvantages of this approach are:

1. Datasets that do not have a query endpoint are ignored. While hundreds of SPARQL endpoints are known to exist today, there are at least thousands of Linked Datasets.
2. Datasets that have a custom API and/or that require an API key in order to pose questions, are not generally accessible and require either appropriation to a specific API or the creation of an account in order to receive a custom key.

3. For practical reasons, otherwise standards-compliant SPARQL endpoints put restrictions on either the number of triples that can be retrieved or the number of rows that can be involved in a sort operation that is required for paginated retrieval.[12] This makes dataset retrieval incomplete.
4. Existing LOD observatories show that SPARQL endpoint availability is quite low.[13] This may be a result of the fact that keeping a SPARQL endpoint up and running requires considerably more resources than hosting a Web document.

Downloading data dumps is the third approach to collecting large volumes of LOD. Its disadvantages are:

1. Datasets that are not available as datadump are ignored.
2. Datasets that have only part of their documents available for download are incomplete.

With the LOD Laundromat we want to clean existing datasets, not create a new dataset that is a collection of parts coming from different datasets (like BTC, for instance). For most datasets for which a SPARQL endpoint exists, we are also able to find a datadump version. We therefore believe that downloading datadumps is the best approach for collecting large amounts of data documents for cleaning.

4 Implementation: the LOD Washing Machine

In the previous section we have describe the requirements that we believe Linked Datasets should fulfill in order to be more useful in practice. We also explained why we have chosen to download datadumps is order to guarantee the best completeness. Here, we will make the aforementioned requirements concrete in such a way that they can be automatically applied to dirty Linked Datasets. The part of the LOD Laundromat that performs automated data cleaning is called the LOD Washing Machine.[14]

Step A: Collect URLs that denote dataset dumps Before we start laundrying data, we need some dirty data to fill our LOD Laundry Basket. The LOD Washing Machine does not completely automate the search for the initial seed points for collecting LOD. The are four reasons for this. Firstly, catalogs that collect metadata descriptions must be accessed by website-specific APIs. Secondly, standards-compliant metadata descriptions are stored at multiple locations and cannot always be found by Web search operations that can be automated. Thirdly, metadata descriptions of datasets, whether standards-compliant

[12] E.g., Virtuoso, an often used triple store, by default limits both the result set size and the number of rows within a sorting operation.

[13] `http://sparqles.okfn.org/`

[14] Code available at `https://github.com/LODLaundry/LOD-Washing-Machine`.

or catalog-specific, are often outdated (e.g., pointing to an old server) or incomplete. Finally, many datasets are not described anywhere and require someone to know the server location where the data is currently stored.

Due to these reasons, the LOD Washing Machine relies on catalog-specific scripts that collect such seed URLs for washing. An example of this is the CKAN API[15], which provides access to the datasets described in the Datahub, including the datasets in the original LOD cloud. This means that URLs that are not included in a LOD catalog or portal are less likely to be washed by the LOD Washing Machine. In addition, we have added several seed points by hand for datasets that we know reside at specific server locations. Anyone can queue washing jobs by adding such seed URLs to the LOD Laundry Basket through the LOD Laundromat Website.

Some URL strings – e.g., values for the "URL" property in a catalog – do not parse according to the RFC 3986 grammar.[16] Some URL strings are parsed as IRIs but not as URLs, mostly because of unescaped spaces. Some URL strings parse per RFC 3986, but have no IANA-registered scheme[17], or the `file` scheme which is host-specific and cannot be used for downloading. The LOD Washing Machine uses only URLs that parse per RFC 3986 (after IRI-to-URL conversion) and that have an IANA-registered scheme that is not host-specific.

Step B: Connect to the hosting server When processing the list of URLs from the previous step, we must be careful with URLs that contain the same authority part, since they are likely to reside at the same server. Since some servers do not accept multiple (near) simultaneous requests from the same IP, we must avoid parallel processing of such URLs. The LOD Washing Machine therefore groups URLs with the same authority and makes sure they get processed in sequence, not in parallel. This is implemented by handling URLs with the same authority in a single thread.

At the level of TCP/IP, not all URL authorities denote a running server or host. Some running servers do not react to requests (neither reject nor accept), and some actively reject establishing a connection. Some connections that are established are broken off during communication.

Step C: Communicate with the hosting server Once a connection has been established over TCP/IP, the LOD Washing Machine sends an HTTP request with SSL verification (for secure HTTP) and an accept header that includes a preference for LOD content types. This includes standardized content types and content types that occur in practice.

Some requests are unsuccessful, receiving either a server, existence, or permission error. Some requests are unsuccessful due to redirection loops.

Step E: Unpack archived data Many Linked Datasets are contained in archives. The LOD Washing Machine supports the archive filters and formats that are

[15] `http://ckan.org/`

[16] `http://tools.ietf.org/html/rfc3986`

[17] `http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml`

supported by library libarchive[18]. The LOD Washing Machine accesses archives in a stream and opens additional streams for every archive entry it contains. Since archive entries can themselves be archives, this procedure is nested, resulting in a tree of streams. The root node of the tree is the stream of the original archive file, the leaf nodes are streams of non-archived files, and the other non-leaf nodes are streams of intermediate archived files.

Some archives cannot be read by libarchive, which throws an exception. We have not been able to open these archives with any of the standard unarchiving tools on Linux. Consequently, the LOD Washing Machine gives up on such archived files, but does report the exception that was thrown.

Step F: Guess serialization format In order to parse the contents of the textual data that resides in the leaf nodes of the stream tree, we need to know the grammar of that data. The LOD Washing Machine uses content types to denote the grammar that is used for parsing. There are various ways in which the content type of a streamed file can be assessed. The most reliable way is to parse the whole file using each of the RDF serialization parsers, and take the one that emits the least syntax errors and/or reads the most valid RDF triples. A theoretical example of why one needs to parse the whole file, not just a first segment of it, can be given with respect to the difference between the Turtle and TriG formats. This difference may only become apparent in the last triple that appears in the file, by the occurrence of curly brackets (indicating a named graph).

Unfortunately, parsing every dataset with every parser is inefficient (CPU) and requires either local storage of the whole file (disk space) or multiple downloads of the same file (bandwidth).

In addition, we make the observation that the standardized RDF serialization formats occur in two families: XML-like (RDF/XML, RDFa) and Turtle-like (Turtle, TriG, N-Triples, N-Quads). The distinction between these two families can be reliably made by only looking at an initial segment of the file.

In order to keep the hardware footprint low, the LOD Washing Machine tries to guess the content type of a file based on a parse of only a first chunk of that file, in combination with the extension of the file (if any) and the content type header in the HTTP response message (if any). Using a look-ahead function on the stream, the LOD Washing Machine can use the first bytes on that stream in order to guess its content type, without consuming those bytes so that no redownload is necessary. The number of bytes available in the look-ahead is the same as the stream chunk size that is used for in-memory streaming anyway.

As explained above, this method may result in within-family mistakes, e.g., guessing Turtle for TriG or guessing N-Triples for N-Quads. In order to reduce the number of within-family mistakes, we use the content type and file extension. If these denote serialization formats that belong within the guessed family, we use that format. Otherwise, we use the most generic serialization format within the guessed family.

[18] `https://code.google.com/p/libarchive/`

This approach ensures that the LOD Washing Machine uses a fully streamed pipeline and relatively few hardware resources.

Step G: Syntax errors while parsing RDF serializations The LOD Washing Machine parses the whole file using standards-conforming grammars. For this it uses the parsers from the SemWeb library [11]. This library passes the RDF 1.1 test cases and is actively used in SW research and applications. Using this library, the LOD Washing Machine is able to recognize different kinds of syntax errors and recover from them during parsing. We enumerate some of the most common syntax errors the LOD Laundromat is able to identify:

- Bad encoding sequences (e.g., non-UTF-8).
- Undefined IRI prefixes.
- Missing end-of-statement characters between triples (i.e., 'triples' with more than three terms).
- Non-escaped, illegal characters inside IRIs.
- Multi-line literals in serialization formats that do not support them (e.g., multi-line literals that are only legal in Turtle, also occur in N-Triples and N-Quads).
- Missing or non-matching end tags (e.g., RDF/XML).
- End-of-file occurrence within the last triple (probably indicating a mistake that was made while splitting files).
- IRIs that no not occur in between angular brackets (Turtle-family).

The LOD Washing Machine reports each syntax error it comes across. For data documents that contain syntax errors, there is no formal guarantee that a one-to-one mapping between the original document and a cleaned sibling document exists. This is an inherent characteristic of dirty data and the application of heuristics in order to clean as many stains as possible. In the absence of a formal model describing all the syntactic mistakes that can be made, recovery from arbitrary syntax errors is more of an art than a science. We illustrate this with the following example:

```
ex:a1  ex:a2  $””  ex:b1  ex:b2  ex:b3  .
                   ex:c1  ex:c2  ex:c3  .
                   ...
                   ex:z1  ex:z2  ex:z3  .
              ”””  .
```

A standards-compliant RDF parser will not be able to parse this piece of syntax and will give a syntax error. A common technique for RDF parsers is to look for the next end-of-triple statement (i.e., the dot at the end of the first line), and resume parsing from there. This results in parsing the collection of triples starting with ⟨`rdf:c1`, `rdf:c2`, `ex:c3`⟩ and ending with ⟨`rdf:z1`, `rdf:z2`, `ex:z3`⟩. The three double quotes that occur at the end of the code sample will result in a second syntax error.

However, using other heuristics may produce very different results. For instance, by using minimum error distance, the syntax error can also be recovered by replacing the dollar sign with a double quote sign. This results in a single triple with a unusually long, but standards-compliant, literal term.

Step H: De-duplicate RDF statements The LOD Washing Machine loads the parsed triples into a memory-based triple store. By loading the triples into a triple store, it performs deduplication of interpreted RDF statements. Deduplication cannot be performed without interpretation, i.e., on the syntax level, because the same RDF statement can be written in different ways. Syntactically, the same triple can look different due to the use of character escaping, the use of extra white spaces, newlines and/or interspersed comments, the use of different/no named prefixes for IRIs, and abbreviation mechanisms in serialization formats that support them (e.g., RDF/XML, Turtle). Another source of the many-to-one mapping between syntax and semantics occurs for RDF datatypes / XML Schema 1.1 datatypes, for which multiple lexical expressions can map onto the same value.[19] For example, the lexical expressions 0.1 and 0.10000000009 map to the same value according to data type `xsd:float`, but to different values according to data type `xsd:decimal`.

While reading RDF statements into the triple store, the contents of different data documents are stored in separate transactions, allowing the concurrent loading of data in multiple threads. Each transaction represents an RDF graph or set of triples, thereby automatically deduplicating triples within the same file.

Step I: Save RDF in a uniform serialization format Once the triples are parsed using an RDF parser, and the resulting RDF statements are loaded into memory without duplicates, we can use a generator of our choice to serialize the cleaned data. We want our generator to be compliant with existing standards, and we want to support further processing of the data, as discussed in section 3.1. The LOD Washing Machine produces data in a canonical format that enforces a one-to-one mapping between data triples and file lines. This means that the end-of-line character can be reliably used in subsequent processing, such as pattern matching (e.g., regular expressions) and parsing. This also means that data documents can be easily split without running the risk of splitting in-between triples. Furthermore, the number of triples in a graph can be easily and reliably determined by counting the number of lines in a file describing that graph. Secondly, the LOD Washing Machine leaves out any header information. This, again, makes it easy to split existing data documents into smaller parts, since the first part of the file is not treated specially due to serialization-specifc header declarations (e.g., RDF/XML, RDFa) and namespace definitions (e.g., RDF/XML, Turtle). Thirdly, the LOD Washing Machine replaces all occurrences of blank nodes with well-known IRIs[20], in line with the RDF 1.1 specification[21]. Effectively, this means that blank nodes are interpreted as Skolem constants, not

[19] `http://www.w3.org/TR/xmlschema11-2/`

[20] `https://tools.ietf.org/html/rfc5785`

[21] `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/\#section-skolemization`

as existentially quantified variables. The Skolem constant is an IRI that is based on the URL that was used to stream the RDF data from, thereby making it a universally unique name at the moment of processing.[22] This makes it easy to append and split data documents, without the need to standardize apart blank nodes that originate from different graphs.

From the existing serialization formats, N-Triples and N-Quads come closest to these requirements. Since the tracking of named graphs is out of scope for our initial version of the LOD laundry (see section 6), we use a canonical form of N-Triples that excludes superfluous white space (only one space between the RDF terms in a triple and one space before the end-of-triple character), superfluous newlines (only one newline after the end-of-triple character), and comments (none at all). Newlines that occur in multi-line literals, supported by some serialization formats, are escaped according to the N-Triples 1.1 specification. Also, simple literals are not written, always adding the XML Schema string datatype explicitly.

Step J: VoID closure After having stored the data to a canonical format, we make use of the fact that the valid triples are still stored in memory, by perform a quick query on the memory store. In this query we derive any triples that describe Linked Datasets. Specifically, we look for occurrences of predicates in the VoID namespace. We store these triples in a publicly accessible metadata graph that can be queried using SPARQL. For each dataset described in VoID triples, we follow links to datadumps (if present), add them to the LOD Laundry Basket, and clean those datadumps by using the LOD Washing Machine as well. Since a dataset may describe a dataset that describes another dataset, this process is recursive.

Step K: Consolidate and disseminate datasets for further processing Since we want to incentivise the dataset creators to improve their adherence to guidelines, we keep track of all the mistakes that were discussed in this section. The mistakes (if any) are asserted together with some basic statistics, e.g. number of triples, number of bytes processed, in the publicly queryable metadata graph. For syntax errors we include the line and column number at which the error occurred, relative to the original file. This makes it easy for the dataset maintainers to improve their data and turn out cleaner in a next wash, since the metadata descriptions are automatically updated at future executions of the LOD Washing Machine.

5 The LOD Laundromat Web service

When the LOD Washing Machine has cleaned a data document, it is ironed and folded and made available on a publicly accessible Website that provides addi-

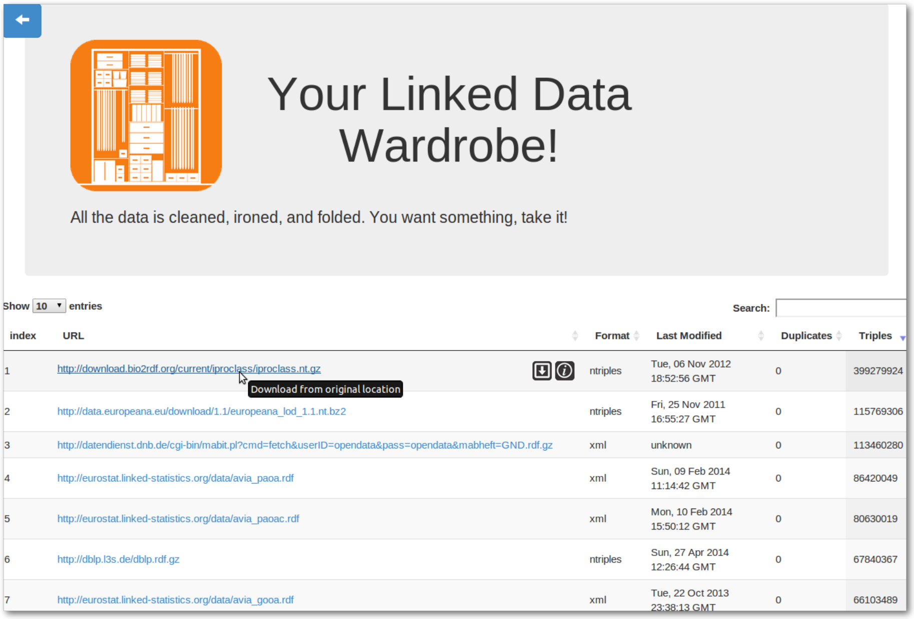
[22] When a new file is disseminated at the same URL at a later point in time, the same Skolem constant may be used to denote a different blank node. Using skolemization, this becomes an instance of the generic problem that IRIs can denote different things at different times, as the data document is updated.

tional support for data consumers. We now describe the components that make up this Website and lift out the support features that make LOD Laundromat a good source for finding large volumes of high-quality Linked Data.

5.1 LOD Wardrobe

The LOD Wardrobe (Figure 1) is where the cleaned datasets are disseminated for human data consumers. The data documents are listed in a table that can be sorted according to various criteria (e.g., cleaning data, number of triples). For every data document, a row in the table includes links to both the old (dirty) and new (cleaned) data files, as well as a button that brings up a pop-up box with all the metadata for that data document. Furthermore, it is easy to filter the table based on a search string and multiple rows from the table can be selected for downloading at the same time.

Fig. 1. The LOD Wardrobe is available at `http://lodlaundromat.org/wardrobe`



Your Linked Data Wardrobe!

All the data is cleaned, ironed, and folded. You want something, take it!

index	URL	Format	Last Modified	Duplicates	Triples
1	http://download.bio2rdf.org/current/iproclass/iproclass.nt.gz	ntriples	Tue, 06 Nov 2012 18:52:56 GMT	0	399279924
2	http://data.europeana.eu/download/1.1/europeana_lod_1.1.nt.bz2	ntriples	Fri, 25 Nov 2011 16:55:27 GMT	0	115769306
3	http://datendienst.dnb.de/cgi-bin/mabit.pl?cmd=fetch&userID=opendata&pass=opendata&mabheft=GND.rdf.gz	xml	unknown	0	113460280
4	http://eurostat.linked-statistics.org/data/avia_paoa.rdf	xml	Sun, 09 Feb 2014 11:14:42 GMT	0	86420049
5	http://eurostat.linked-statistics.org/data/avia_paoac.rdf	xml	Mon, 10 Feb 2014 15:50:12 GMT	0	80630019
6	http://dblp.l3s.de/dblp.rdf.gz	ntriples	Sun, 27 Apr 2014 12:26:44 GMT	0	67840367
7	http://eurostat.linked-statistics.org/data/avia_gooa.rdf	xml	Tue, 22 Oct 2013 23:38:13 GMT	0	66103489

5.2 SPARQL endpoint

All the metadata collected during the cleaning process is stored in an RDF graph that is publicly accessible through the SPARQL endpoint `http://lodlaundromat.`

`org/sparql`. For human data consumers we deploy the feature-rich SPARQL editor Yasgui.[23] For machine consumption the SPARQL endpoint can be queried algorithmically. For instance, a SPARQL query can return URLs for downloading all clean data documents with over one million syntactically correct triples. In this way LOD Laundromat provides a very simple interface for running Big Data experiments. The metadata stored by the LOD Washing Machine includes information such as the number of triples in a dataset:

- the number of removed duplicates,
- the original serialization format,
- any VoID descriptions that were found,
- various kinds of syntax errors,
- and more.

The metadata that the LOD Wardrobe publishes is continuously updated whenever new cleaned laundry comes in.

5.3 Visualizations

Besides access to the datasets, the LOD Laundromat also provides real-time visualizations of the crawled data. These visualizations are small JavaScript widgets that use SPARQL queries on the metadata SPARQL endpoint.

Purely for illustrative purposes, we include a snapshot of such a widget in Figure 2. For a collection of 1.276 cleaned documents (containing approximately 2 billion triples) this widget shows the serialization format that was used to parse the original file. The majority of documents from this collection, 59.2%, are serialized as RDF/XML. Turtle and RDFa amount to 29.5% and 6.7% respectively. Only 4.4% of all documents are serialized as N-Triples.

As another example of the kinds of queries that can be performed on the SPARQL endpoint, we take the HTTP `Content-Length` header. Values for this header are often set incorrectly. Ideally, a properly set `Content-Length` header would allow data consumers to retrieve data more efficiently, e.g., by load-balancing data depending on the expected size of the response. However, our results show that 32% of the documents return an invalid content length value, thereby showing that in practice it is difficult to reliably make use of this property.

5.4 LOD Laundry Basket

In order to extend our collection of datasets over time, users can add seed URLs to the LOD Laundry Basket. Seed points can be either URLs that point to VoID descriptions, or to data dumps directly. Seed locations can be added through a Web form or through a direct HTTP GET request.

[23] `http://yasgui.laurensrietveld.nl/`

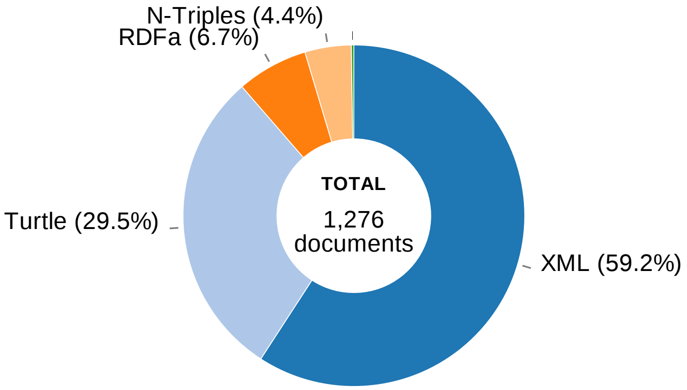


Fig. 2. RDF serialization formats for a collection of RDF documents. Illustrative example of a visualization widget at `http://lodlaundromat.org/visualizations`.

6 Conclusion

Existing research shows that many LOD does not comply with existing standards. To deal with this issue, we have presented LOD Laundromat, a uniform way of publishing other peoples dirty data. Using LOD Laundromat, we publish standards- and guidelines-compliant datasets that are siblings of existing, idiosyncratic datasets. LOD Laundromat implements a Linked Data cleaner that *continuously* crawls for additional datasets; the amount of data that we publish (over ten billion triples at the time of writing) already surpasses that of existing data collections, such as the Billion Triple Challenge. In addition, the LOD Laundromat publishes metadata for every cleaned document on a publicly accessable Web site, and through machine-accessable Web services. Because anybody can drop their dirty data in the LOD Laundry Basket, the coverage of the LOD Laundromat will increase over time. All datasets are published in a very simple canonical form of N-Triples, which makes it easy for post-processing tools to parse, possibly in streamed form. By using the LOD Laundromat, data consumers do not have to worry about different serialization formats, syntax errors, encoding issues, or triple duplicates. In doing so, LOD Laundromat can act as an enabler for Big Data and SW research, as well as a provider of data for Web-scale applications.

Although the LOD Laundromat offers many advantages for data consumers today, we aim to further increase the level of support. Fistly, the metadata we collect does not yet make use of existing vocabularies, like DCAT[24], VoID, and Prov-O[25]. Secondly, LOD Laundromat currently disseminates datasets in the N-Triples serialization format, in which it is not possible to represent multiple graphs. Even though the use of multiple graphs within the same data document is

[24] `http://www.w3.org/TR/vocab-dcat/`

[25] `http://www.w3.org/TR/prov-o/`

not very common today, the few datasets for which this is used would be better supported by the N-Quads format. This also requires the scope of the triple deduplication phase to be narrowed down to graphs. Thirdly, not all Linked Data is Open. Some data may be licensed under conditions that do not allow free data reuse. However, restricting licenses are difficult to detect by automated means, since very few datasets contain explicit licensing conditions. Still, in case a dataset does explicitly mention a license that is not defined open by the Open Data Commons[26], we would like the LOD Washing Machine to skip it. Finially, we may choose to store multiple versions of the collection of cleaned datasets as different 'snapshots'. Such snapshots may, for instance, improve the reproducibility of LOD experiments.

References

1. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a Collaboratively Created Graph Database for Structuring Human Knowledge . In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data . pp. 1247–1250. ACM (2008)
2. Callahan, A., Cruz-Toledo, J., Ansell, P., Dumontier, M.: Bio2rdf Release 2: Improved Coverage, Interoperability and Provenance of Life Science Linked Data . In: The Semantic Web: Semantics and Big Data, pp. 200–212. Springer (2013)
3. Ermilov, I., Martin, M., Lehmann, J., Auer, S.: Linked Open Data Statistics: Collection and Exploitation. In: Knowledge Engineering and the Semantic Web, pp. 242–249. Springer (2013)
4. Hogan, A., Harth, A., Passant, A., Decker, S., Polleres, A.: Weaving the pedantic web. In: Linked Data on the Web Workshop (2010)
5. Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S.: An Empirical Survey of Linked Data Conformance. Web Semantics: Science, Services and Agents on the World Wide Web 14, 14–44 (2012)
6. Käfer, T., Abdelrahman, A., Umbrich, J., O'Byrne, P., Hogan, A.: Observing Linked Data Dynamics. In: The Semantic Web: Semantics and Big Data. Springer (2013)
7. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: A not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data . pp. 1099–1110. ACM (2008)
8. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: a Document-Oriented Lookup Index for Open Linked Data . International Journal of Metadata, Semantics and Ontologies 3(1), 37–52 (2008)
9. Stadtmüller, S., Harth, A., Grobelnik, M.: Accessing Information about Linked Data Vocabularies with vocab.cc . In: Semantic Web and Web Science, pp. 391–396. Springer (2013)
10. Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., Decker, S.: Sig.ma: Live Views on the Web of Data. Web Semantics: Science, Services and Agents on the World Wide Web 8(4), 355–364 (2010)
11. Wielemaker, J.: SWI-Prolog Semantic Web Library 3.0, `http://prolog.cs.vu.nl/download/doc/semweb.pdf`

[26] An overview of machine-readable licence descriptions for data can be found at `http://licenses.opendefinition.org/licenses/groups/all.json`